VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2010 Tomáš Fluksa

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky a výpočetní techniky

Rozhodování pravdivosti formulí Presburgerovy aritmetiky pomocí konečných automatů

Deciding Truth of Formulas of Presburger Arithmetic Using Finite Automata

2010 Tomáš Fluksa

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Fluksa**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Rozhodování pravdivosti formulí Presburgerovy aritmetiky pomocí konečných automatů
Deciding Truth of Formulas of Presburger Arithmetic Using Finite Automata

Zásady pro vypracování:

Presburgerova aritmetika je logikou prvého řádu, jejímž zamýšleným modelem je množina celých (resp. přirozených) čísel. Ve formulích je možno používat operátor sčítání, nikoliv však násobení. Na rozdíl od aritmetiky celých čísel je pravdivdost formulí Presburgerovy aritmetiky rozhodnutelná.

1. Nastudujte příslušnou problematiku.
2. Implementujte algoritmus pro rozhodování pravdivosti formulí Presburgerovy aritmetiky založený na použití konečných automatů.
3. Otestujte váš program na vhodně zvolených příkladech.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Zdeněk Sawa, Ph.D.**

Datum zadání: 30.11.2008

Datum odevzdání: 07.05.2010

doc. Dr.Ing. Eduard Sojka
vedoucí katedry

prof. Ing. Ivo Vondrák, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 7. 5. 2010 ……..……

podpis

Poděkování

Tímto bych rád poděkoval Ing. Zdeňku Sawovi za ochotné a obětavé vedení a jeho velmi cenné rady a připomínky, které mi velice pomohly k dokončení této práce.

Abstrakt

Tato bakalářská práce se zabývá možností rozhodovat pravdivost formulí Presburgerovy aritmetiky pomocí konečných automatů. V první části je obecný popis Presburgerovy aritmetiky a specifikace problému. Další kapitola se zabývá nastudováním konečných automatů, operací nad těmito automaty a jejich využitím k dosažení cíle této práce. V poslední části je popis vytvořeného počítačového programu, od načtení formule přes její zpracování pomocí konečných automatů po konečné vyhodnocení pravdivosti.

Klíčová slova

Presburgerova aritmetika, deterministický konečný automat, nedeterministický konečný automat, lexikální analýza, syntaktická analýza, Java

Abstract

The bachelor thesis deals with possibility to decide truth of formulas of Presburger arithmetic using finite automata. In first chapter is general description of Presburger arithmetic and specification of problem. Next chapter goes in for study of finite automata, operations over these automata and their utilization for reaching aim of this work. Last chapter describes the computer program, from reading input formula through its processing by using finite automata to final evaluation of truth.

Keywords

Presburger arithmetic, deterministic finite automaton, nondeterministic finite automaton, lexical analysis, syntactic analysis, Java

Obsah

Seznam obrázků

1. Úvod

Presburgerova aritmetika je logikou prvého řádu, jejímž zamýšleným modelem je množina celých (resp. přirozených) čísel. Ve formulích je možno používat operátor sčítání, nikoliv však násobení. Na rozdíl od aritmetiky celých čísel je pravdivost formulí Presburgerovy aritmetiky rozhodnutelná, což dokázal ve 20. létech 20. století polský matematik Mojžesz Presburger s využitím tzv. metody eliminace kvantifikátorů. Následně bylo snahou ukázat podobný algoritmus i s připuštěním predikátu pro násobení. Nemožnost tohoto úkolu ukázal později Kurt Gödel. Mnohem později bylo ukázáno, že každý algoritmus, řešící problém ThAdd (problém pravdivosti teorie sčítání) má pro formuli délky n složitost minimálně 2^{2^n}. [1]

Cílem bakalářské práce je nastudování problematiky rozhodování pravdivosti formulí Presburgerovy aritmetiky pomocí konečných automatů (KA), implementace algoritmu a otestování na vhodně zvolených příkladech. Cílem je tedy sestrojit takový KA, který pracuje nad touto aritmetikou, jeho vstupem je formule a výstupem je hodnota říkající, zda je daná vstupní formule tímto automatem přijata (zda je pravdivá). Je tedy třeba sestrojit automaty pracující nad touto abecedou aritmetických formulí:

$$\{\forall, \exists, \neg, \wedge, \vee, (,), x, +, \Rightarrow, \Leftrightarrow\},$$

kde $\neg$, $\wedge$ a $\vee$ jsou logické operátory, „(“ a „)“ závorky, $\forall, \exists$ jsou kvantifikátory, x je symbol používaný k definování proměnných (nebo také číselných konstant z oboru přirozených čísel), + je operátor sčítání, $\Rightarrow$ je implikace, a $\Leftrightarrow$ ekvivalence. Dále jsou definovány následující relace nad proměnnými:

$$\{x = y, x = x, x + y = z, x + x = y, x > y, x \geq y, x < y, x \leq y, x > x, x \geq x, x < x, x \leq x\}.$$

Tyto relace tvoří atomické formule, lze o nich říct, že jsou složeny následovně:

<term> <relační operace> <term>

Každý z těchto termů může být složen z libovolného počtu operací součtu mezi proměnnými a číselnými konstantami. Formule jsou definovány následujícím způsobem [2]:

1. ϕ , kde ϕ je atomická formule,
2. $\phi_1 \wedge \phi_2, \phi_1 \vee \phi_2, \neg\phi_1, \phi_1 \Rightarrow \phi_2, \phi_1 \Leftrightarrow \phi_2$ kde ϕ_1 a ϕ_2 jsou menší formule,
3. $\exists x_i[\phi_1], \forall x_i[\phi_1]$, kde ϕ_1 je menší formule.

Cílem je tedy postupně pro jednotlivé atomické formule vytvořit KA, z nichž dále za použití operací nad těmito automaty (průniku, sjednocení, doplňku, nedeterminismu a následného převodu zpět na deterministický konečný automat) získáme výsledný deterministický konečný automat (DKA) říkající, zda je formule platná, či nikoliv.

2. Konečné automaty

2.1 Deterministický konečný automat

Deterministický konečný automat (zkráceně DKA) je pětice $(Q, \Sigma, \delta, q_0, F)$, kde:

- Q je konečná množina stavů
- Σ je konečná množina vstupních symbolů, nazývaná abeceda,
- $\delta : Q \times \Sigma \rightarrow Q$ je tzv. přechodová funkce,
- $q_0 \in Q$ je počáteční stav a
- $F \subseteq Q$ je množina přijímajících stavů.

Na počátku se automat nachází v definovaném počátečním stavu q_0. Přečte jeden symbol ze vstupu, který je definován v abecedě $\sum$ a podle definované přechodové funkce $\delta : Q \times \Sigma \rightarrow Q$ přejde do stavu, který odpovídá aktuálnímu stavu a přečtenému symbolu. Tento postup se opakuje tak dlouhou, dokud jsou na vstupu symboly ke čtení. Po přečtení posledního symbolu je slovo přijato tehdy, pokud se aktuální stav nachází v množině přijímacích stavů $F \subseteq Q$. V opačném případě přijato není. Množina všech řetězců, které daný automat přijme, tvoří regulární jazyk [3]

DKA má definovanou přechodovou funkci u všech stavů pro všechny symboly z abecedy Σ, kdy každému symbolu odpovídá právě jeden stav.

2.2 Nedeterministický konečný automat

Nedeterministický konečný automat (zkráceně NKA) je uspořádaná pětice $(Q, \Sigma, \delta, S, F)$, kde [3]:

- Q je konečná množina stavů,
- Σ je konečná abeceda,
- $\delta: Q \times \Sigma \rightarrow \mathrm{P}(\mathrm{Q})$ je přechodová funkce,
- $S \subseteq Q$ je množina počátečních stavů a
- $F \subseteq Q$ je množina přijímacích stavů.

Odlišnosti NKA oproti DKA:

- v aktuálním stavu máme možnost přechodu do více různých stavů, nebo nemusíme mít možnost přechodu do jiného stavu

- NKA umožňuje i tzv. ε-přechody, kdy přecházíme do následujícího stavu bez čtení vstupního znaku

- je povoleno více počátečních stavů oproti DKA, který má právě jeden.

2.3 Využití konečných automatů na binární reprezentaci proměnných

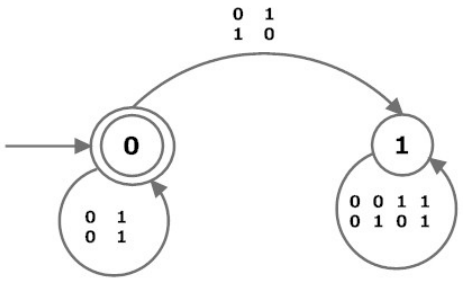
Abeceda, kterou automat rozpoznává, smí být prakticky cokoliv. Může se jednat o samotné znaky abecedy, čísla nebo uspořádané n-tice znaků. Pokud seřadíme proměnné a čísla vyskytující se v aritmetické formuli pod sebe, jejich binární zápis čtený po sloupcích může sloužit jako vstupní abeceda. Pro *i* proměnných tedy bude abeceda vypadat následovně [2]:

$$\sum_{\mathrm{i}} = \left\{ \begin{bmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \cdot \\ \cdot \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ \cdot \\ \cdot \\ 1 \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 1 \\ \cdot \\ \cdot \\ 1 \\ 1 \end{bmatrix} \right\}$$

Obr. 1: vstupní abeceda DKA

Pro *i* proměnných bude mít abeceda celkem i^2 znaků, neboť je třeba pracovat se všemi možnými binárními kombinacemi vstupních proměnných. Když použijeme abecedu $\sum_i$ ve spojení s automatem, můžeme jej vyjádřit jako $\left(Q,\ 2^{\Sigma \mathrm{i}} \cup \left(\{0,1\} \times 2^{\Sigma \mathrm{i}}\right), \delta, q_0, F\right)$, kde $2^{\Sigma \mathrm{i}} \cup \left(\{0,1\} \times 2^{\Sigma \mathrm{i}}\right)$ je abeceda všech znaků [4].

Není těžké si představit, jak by měl vypadat automat přijímací 2 čísla *x, y právě* tehdy, pokud si budou rovna. Vytvoříme si počáteční stav, který bude zároveň přijímací. Při načtení dvojice nul nebo jedniček automat v tomto stavu zůstává. Jakmile ale načte jinou libovolnou dvojici, přechází do dalšího stavu, který si pojmenujeme 1. V tomto stavu již pak zůstává až do konce načítání vstupu pro všechny znaky. Tímto postupem lze snadno vytvořit dvoustavový automat, který přijme zadané slovo pouze tehdy, jsou-li si obě proměnné na vstupu rovny. Odpovídající graf, reprezentující tento automat společně s přechodovou tabulkou budou vypadat následovně:

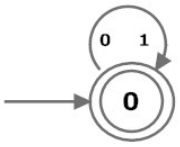


x	**0**	**0**	**1**	**1**
y	**0**	**1**	**0**	**1**
↔0	0	1	1	0
1	1	1	1	1

Obr. 2: Graf a přechodová tabulka automatu pro relaci x = y

U grafu se počáteční stav značí vstupní šipkou, přijímací dvojitým kolečkem. V tabulce šipka směřující vpravo značí, že se jedná o počáteční stav q_0, šipka směřující vlevo označuje přijímací stav patřící do množiny přijímacích stavů F. Pokud šipka směřuje oběma směry, jde o stav počáteční a zároveň přijímací.

Zdá se, že toto je nejjednodušší automat pro základní matematickou operaci, rovnost. Opak je však pravdou. V případě, že budeme chtít rozhodnout, jestli $(x = x)$, budeme mít abecedu $\sum_i = \{[0], [1]\}$. Tedy, $i = 1$. Nyní si již vystačíme s automatem, který bude mít jediný stav. Číslo se totiž vždy rovná samo sobě:

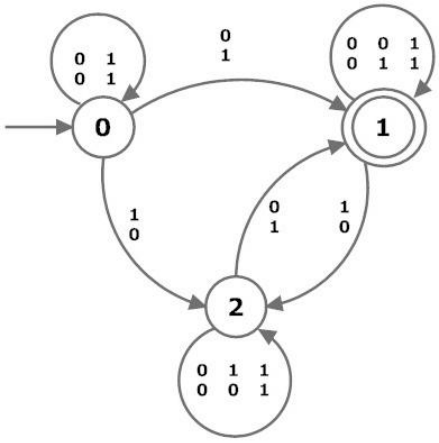


x	0	1
↔0	0	0

Obr. 3: Graf a přechodová tabulka automatu pro výraz x = x

Tento automat je tím pádem shodný i pro relace $(x \leq x, x \geq x)$. Pro relace $(x < x, x > x)$ bude totožný s jediným rozdílem: stav 0 již nebude v množině $F \subseteq Q$. Lze vytvořit nový automat, nebo použít předchozí a provést doplněk, čímž se z přijímacího stavu stane stav, který není přijímací.

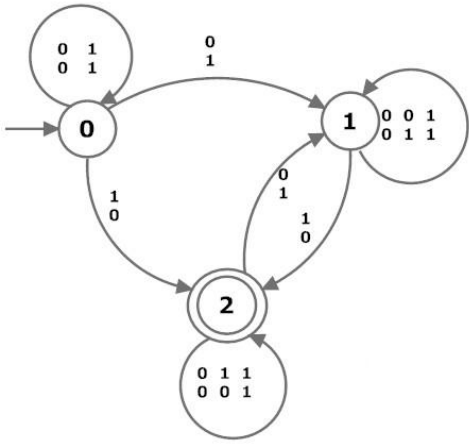
Sestrojení automatu pro relaci $x < y$ je již o něco náročnější. Počáteční stav nebude přijímací, pokud načteme dvojici jedniček nebo nul, zůstáváme v tomto stavu. Při načtení dvojice $\begin{bmatrix}0\\1\end{bmatrix}$ je splněna podmínka, automat tedy přechází do přijímacího stavu, který si pojmenujeme 1. Naopak pokud bude na vstupu dvojice $\begin{bmatrix}1\\0\end{bmatrix}$, potřebujeme další stav, který nebude přijímací. Tento pojmenujeme jako 2. Mezi stavy 1 a 2 pak bude automat přecházet podle toho, zda bude podmínka splněna nebo ne.



x	0	0	1	1
y	0	1	0	1
→0	0	1	2	0
←1	1	1	2	1
2	2	1	2	2

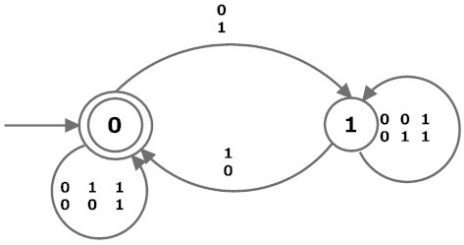
Obr. 4: Graf a přechodová tabulka automatu pro relaci x < y

Automaty pro relace $(x > y, x \geq y,\ x \leq y)$ lze sestrojit podobným způsobem:



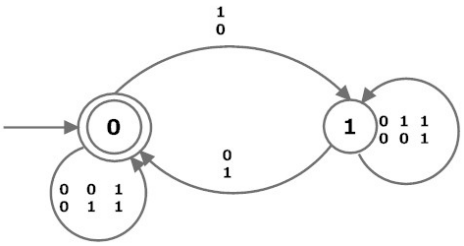
x	**0**	**0**	**1**	**1**
y	**0**	**1**	**0**	**1**
→0	0	1	2	0
1	1	1	2	1
←2	2	1	2	2

Obr. 5: Graf a přechodová tabulka automatu pro relaci x > y



x	**0**	**0**	**1**	**1**
y	**0**	**1**	**0**	**1**
↔0	0	1	0	0
1	1	1	0	1

Obr. 6: Graf a přechodová tabulka automatu pro relaci x ≥ y



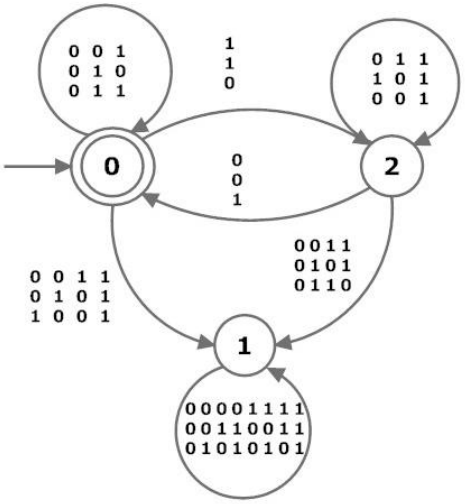
x	**0**	**0**	**1**	**1**
y	**0**	**1**	**0**	**1**
↔0	0	0	1	0
1	1	0	1	1

Obr. 7: Graf a přechodová tabulka automatu pro relaci x ≤ y

To by byly automaty pro základní relace. Dále je potřeba ještě 2 základní automaty: součet 2 různých proměnných je roven třetí $(x + y = z)$ a součet 1 proměnné se stejnou proměnnou je roven druhé $(x + x = y)$. Jak si později ukážeme, tyto dva a výše uvedené automaty nám budou stačit. Je totiž zbytečné složitě vymýšlet automaty pro složitější relace a operace sčítání, když můžeme využít operací průniku a sjednocení automatů. Tento postup má snad jedinou nevýhodu – velikost výsledného automatu. Pokud totiž sestrojíme automat přímo pro co největší část formule, jsme ho schopni sestrojit tak, aby byl skutečně minimální, bez

zbytečných stavů. Zatímco, když rozložíme stejný vstup na výše uvedené automaty, následným použitím operací průniku či sjednocení v kombinaci s nedeterministickým hádáním pro některou z proměnných může množina stavů Q několikanásobně vzrůst. Oba automaty budou přesto přijímat stejné slovo.

U automatů, které již zahrnují sčítání musíme počítat s přenosy do vyšších řádů. Začneme načítat vstup od nejnižšího řádu a pokaždé když dojde k přenosu bitu do vyššího řádu, musíme v následujícím stavu kontrolovat, jestli si jsou čísla rovny. I tak si u automatu pro výraz $(x + y = z)$ vystačíme se 3 stavy. Počáteční stav bude zároveň stavem přijímacím, druhý stav bude sloužit pro přenosy do vyšších řádů, z něj bude zpětně dosažitelný přijímací stav. A třetí stav, který nebude přijímací a nebude z něho návratu zpět žádným ze symbolů. Do tohoto stavu se přejde, jakmile dojde k nerovnosti. Automat vypadá následovně:

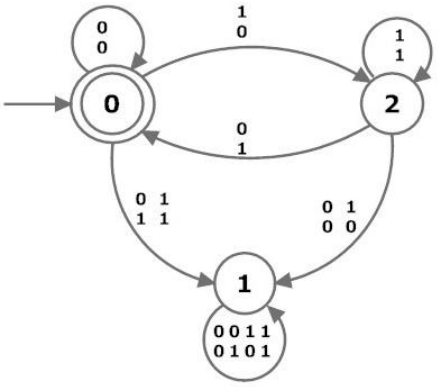


x	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
y	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
z	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
↔0	0	1	1	0	1	0	2	1
1	1	1	1	1	1	1	1	1
2	1	0	2	1	2	1	1	2

Obr. 8: Graf a přechodová tabulka automatu pro výraz x + y = z

Je možnost automaty transformovat tak, aby četly vstup zleva, čili od nejvyššího bitu. Postup by byl takový, že by se otočily šipky označující přechody mezi stavy, odstranily by se nedosažitelné stavy a vzniklé ε-přechody. Výsledkem by byl NKA, který bychom následně převedli na odpovídající DKA. My však ponecháme vytvořené automaty tak, jak jsou.

Poslední z atomických výrazů, které potřebujeme vyjádřit automatem je výraz $(x + x = y)$. V tomto výrazu se vyskytují 2 stejné proměnné, i je tedy rovno 2: $\begin{bmatrix} x \\ y \end{bmatrix}$. Automat bude mít stejný počet stavů jako ten předchozí, avšak již pracuje s jinou abecedou $\sum_i$. Opět budeme potřebovat stav pro přenos bitů do vyšších řádů a jeden chybový stav. Automat bude velmi podobný tomu předchozímu:



x	**0**	**0**	**1**	**1**
y	**0**	**1**	**0**	**1**
↔**0**	0	1	2	1
1	1	1	1	1
2	1	0	1	2

Obr. 9: Graf a přechodová tabulka automatu pro výraz x + x = y

2.4 Operace nad konečnými automaty

S konečnými automaty můžeme provádět operace průniku, sjednocení a doplňku. Pokud máme dva automaty $A_1 = \{Q_1, \sum, \delta_1, q_{01}, F_1\}$ a $A_2 = \{Q_2, \sum, \delta_2, q_{02}, F_2\}$ definované nad stejnou abecedou $\sum$, jejich spojením dostaneme automat $A = \{Q, \sum, \delta, q_0, F\}$, [5], kde:

- $Q = Q_1 \times Q_2$
- $q_0 = (q_{01,}q_{02})$
- $F \subseteq Q$
- $\delta: (Q_1, Q_2) \times \sum \rightarrow (Q_1, Q_2)$
- $\delta: ((q_a, q_b), s) \rightarrow (\delta_1(q_a, s), \delta_2(q_b, s)), q_a \in Q_1, q_b \in Q_2$.

2.4.1 Průnik dvou DKA

Při průniku dvou automatů je množina koncových stavů dána takto: $(q_a, q_b) \in F \iff (q_a \in F_1) \wedge (q_b \in F_2)$. Takovýto automat rozpoznává $L(A_1) \cap L(A_2)$. Počet stavů takto nově vzniklého DKA může být v nejhorším případě až $|Q_1| \cdot |Q_2|$, kde $|Q_1|, |Q_2|$ jsou počty stavů v jednotlivých DKA. Postup pro průnik dvou automatů tedy vypadá takto [5]:

1. Vezmi přijímací stavy $(q_{01,}q_{02})$, a do Q zapiš nový stav odpovídající této dvojici.

2. Načti pro stav $(q_{01,}q_{02})$ přechody pro všechny prvky z $\sum$. Když není takto načtený stav (q_a, q_b) definovaný mezi stavy v množině Q, zapiš tuto dvojici jako nový stav.

3. Pro každý nově vytvořený stav (q_a, q_b) načti všechny přechody $(\delta_1(q_a, s), \delta_2(q_b, s))$. Každý nově vzniklý stav (q_a, q_b), který ještě není definovaný mezi stavy množiny Q, přidej na konec této množiny.

- Opakuj bod 3 tak dlouho, dokud nezpracuješ všechny stavy v Q.

4. Označ všechny stavy (q_a, q_b) jako přijímací právě tehdy, když q_a je v množině přijímacích stavů F_1 a zároveň q_b je v množině přijímacích stavů F_2.

Zmíněný postup lze ilustrovat na příkladu průniku dvou DKA:

x	0	0	1	1
y	0	1	0	1
↔0	0	0	1	0
1	1	0	2	1
2	1	1	2	1

x	0	0	1	1
y	0	1	0	1
→0	0	0	1	1
←1	1	0	2	1
←2	1	2	2	2

x	0	0	1	1
y	0	1	0	1
→0,0	0,0	0,0	1,1	0,1
1,1	1,1	0,0	2,2	1,1
←0,1	0,1	0,0	1,2	0,1
2,2	1,1	1,2	2,2	1,2
1,2	1,1	0,2	2,2	1,2
←0,3	0,1	0,2	1,2	0,2

x	0	0	1	1
y	0	1	0	1
→0	0	0	1	2
1	1	0	3	1
←2	2	0	4	2
3	1	4	3	4
4	1	5	3	4
←5	2	5	4	5

Obr. 10: Průnik dvou DKA

Na obrázku č. 10 lze na prvním řádku vidět 2 odlišné DKA, pracující se stejnou abecedou. Vlevo dole je jejich průnik, napravo poté ten samý automat se stavy pojmenovanými vzestupně od nuly.

2.4.2 Sjednocení dvou DKA

Při sjednocení dvou automatů je množina koncových stavů dána takto: $(q_a, q_b) \in F \iff (q_a \in F_1) \vee (q_b \in F_2)$. Takovýto automat rozpoznává $L(A_1) \cup L(A_2)$. Postup pro sjednocení dvou automatů je totožný, rozdíl je pouze v označení přijímacích stavů. Zatímco u sjednocení musí být oba (q_a, q_b) stavy v množině přijímacích stavů F u původních DKA, u sjednocení stačí, pokud alespoň jeden z těchto stavů byl přijímací. Po průniku automatů z předchozího příkladu by tedy bylo v množině F všech 6 stavů výsledného automatu [5].

2.4.3 Doplněk DKA

Množina přijímacích stavů F obsahuje stavy, které původně nebyly považovány za přijímací, původní přijímací stavy jsou naopak z této množiny vyřazeny. Platí tedy vztah: $q_a \in F_d \iff q_a \notin F$ [5].

2.4.4 Převod NKA na ekvivalentní DKA

Pro každý NKA lze zkonstruovat ekvivalentní DKA, který rozpoznává stejný jazyk, ovšem mnohdy se značným nárůstem stavů. NKA $A = (Q, \sum, \delta, S, F)$ lze převést na ekvivalentní DKA $A = \{Q', \sum, \delta', q_0, F'\}$, kde:

- $Q' = P(Q)$
- $\delta' = P(Q) \times \sum \longrightarrow P(Q)$, kde $\delta(A, s) = U_{q \in A}\, \delta(q, s)$
- $q_0 = S$
- $F' = \{A \in P(Q), A \cap F \neq \emptyset\}$

Na tento převod lze použít podmnožinovou stromovou konstrukci [5] s následujícím postupem:

1. Do kořenového uzlu q_0 vlož obsah množiny S.
2. Pro každý vstupní symbol $s \in \sum$ vytvoř z předchozího uzlu q_0 novou větev do nového uzlu q_i, který bude obsahovat všechny stavy, do kterých může být NKA převeden pomocí vstupního symbolu s.
    - Pokud není pro vstupní symbol definován přechod ani jeden stav z daného uzlu, vytvoř prázdnou množinu, značenou $\{\}$. Pro každé $s \in \sum$ pak stav přechází sám do sebe.
    - Pokud je vytvořená skupina již definována, je označena stejně a dále se již nedělí.
    - Do množiny F' vlož uzel tehdy, pokud obsahuje alespoň jeden stav z množiny F.
3. Postup z bodu 2 opakuj pro každý nově vytvořený uzel, s výjimkou již definovaných.

Výše uvedený postup lze ilustrovat na následujícím převodu NKA na DKA:

	0	1
↔ **0**	-	1,2
↔ **1**	1,2	2
2	0	-

	0	1
↔{**0**,**1**}	{1,2}	{1,2}
{**1**,**2**}	{0,1,2}	{2}
←{**0**,**1**,**2**}	{0,1,2}	{1,2}
{**2**}	0	{}
←{**0**}	{}	{1,2}
{}	{}	{}

	0	1
↔**0**	1	1
1	2	3
←**2**	2	1
3	4	5
←**4**	5	1
5	5	5

Obr. 11: Převod NKA na odpovídající DKA

Nalevo je NKA se 2 počátečními stavy a 2 nedefinovanými přechody. Tento automat lze převést na DKA uprostřed. Vpravo je poté totožný DKA se stavy přejmenovanými vzestupně od nuly.

2.5 Využití konečných automatů pro řešení aritmetických formulí

S využitím výše zmíněných operací nad konečnými automaty jsme již schopni rozhodnout o pravdivosti jednoduchých aritmetických formulí z oboru $\mathbb{N}_0 = \{0,1,2,3,\dots\}$, jako například, zda $(2+3=5)$. Nebo i o něco „složitější“ $(2+3=5) \wedge (2>3)$. Postup bude následující: DKA pro první výraz si označíme jako DKA_1 a DKA pro druhý výraz budeme značit jako DKA_2. Abeceda pro DKA_1 vypadá následovně:

$$\begin{bmatrix}2\\3\\5\end{bmatrix} \sum_i = \left\{ \begin{bmatrix}0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\end{bmatrix}, \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}1\\0\\1\end{bmatrix}, \begin{bmatrix}1\\1\\0\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix} \right\}$$

Můžeme rozhodnout o pravdivosti prvního výrazu, o pravdivosti druhého výrazu také. Avšak zatím nemůžeme provést průnik těchto automatů, neboť DKA_2 pracuje s jinou abecedou, než $\sum_i$. To se dá snadno napravit tím, že rozšíříme abecedu daného automatu o bity pro číslo 5. Počet stavů zůstává stejný, počet znaků abecedy vzroste dvojnásobně. U každého stavu se budou opakovaně vyskytovat stejné přechody pro shodující se dvojice bitů reprezentující čísla $\begin{bmatrix}2\\3\end{bmatrix}$. Upravený DKA_2 vypadá následovně:

2	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
3	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
5	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
→**0**	0	0	1	1	2	2	0	0
1	1	1	1	1	2	2	1	1
←**2**	2	2	1	1	2	2	2	2

Obr. 12: DKA s rozšířenou abecedou

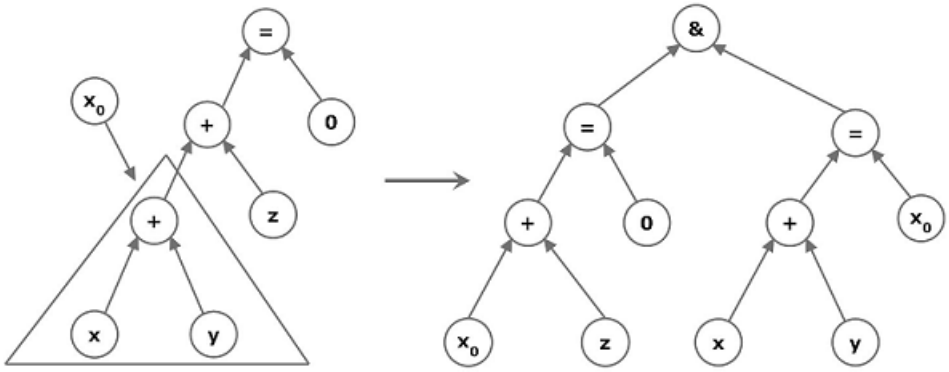
Takto upravený DKA_2 již pracuje nad stejnou abecedou, jako DKA_1, můžeme tedy provést průnik těchto automatů. Vstupní slovo vytvoříme bitovým zápisem jednotlivých čísel pod sebou tím způsobem, že každé číslo převedeme na jeho bitovou reprezentaci, zarovnáme vpravo a chybějící místa vyplníme nulami. Vznikne nám následující vstupní slovo:

$$\begin{matrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{matrix}$$

Obr. 13: Bitová reprezentace vstupu

Toto slovo přečteme zprava doleva (DKA jsou konstruovány pro čtení slova od nejnižších bitů) a po přečtení celého slova můžeme rozhodnout o pravdivosti této formule nahlédnutím do množiny F, zda se v ní aktuální stav vyskytuje, či nikoliv.

V případě, že je term složen z více součtů, je třeba provést další úpravy. Je totiž nemožné vytvořit z hlavy automaty pro všechny různé kombinace součtů. Byla by to zdlouhavá, náročná a navíc zbytečná práce, jelikož použitím určitých úprav můžeme vytvořit výsledný DKA pro jakoukoliv formuli odpovídající specifikaci dané aritmetiky. Pokud máme například atomickou formuli ve tvaru: $(x + y + z = 0)$, můžeme část $(x + y)$ vytknout a nahradit novou proměnnou. Bude tedy platit: $(x + y = x_0)$, po dosazení: $(x_0 + z = 0) \wedge (x + y = x_0)$. Tyto operace již známe a můžeme tedy za použití předchozích znalostí snadno vytvořit výsledný DKA. Tento postup lze názorně zobrazit pomocí následujícího stromu:



Obr. 14: Rozklad složitějšího termu na více operací

Obecně tedy lze atomickou formuli, kde se v termu vyskytuje n proměnných a $n-1$ operací součtu převést na logickou formuli, kde levá atomická formule bude mít v termech jedinou proměnnou. Na pravé straně logické formule je $n-1$ dalších vnořených logických formulí rozkládajících term na n operací součtu.

Pokaždé, když už nebudeme nově vytvořenou proměnnou nahrazující jednu operaci součtu potřebovat, je z abecedy $\sum$ odstraněna. Další úprava spočívá v upravení formule na tvar: $\exists x_0\big((x_0 + z = 0) \wedge (x + y = x_0)\big)$. Postup odstranění proměnné a následná úprava automatů jsou popsány na následujících řádcích.

Pokud převádíme DKA s abecedou $\sum_i$ na konečný automat s abecedou $\sum_{i-1}$, dochází k převodu původního DKA na NKA. Tímto odstraněním proměnné totiž dochází k vytvoření nové abecedy, kdy se v ní každý znak vyskytuje dvakrát. Tím jsou u všech stavů definovány 2 přechodové funkce pro každý znak. Tento automat již není DKA, ale NKA. Ten lze opět převést na DKA algoritmem popsaným v kapitole 2.4.4.

Po odstranění proměnné a následném převodu na DKA je třeba provést ještě jednu úpravu. Při nedeterministickém hledání řešení pro odstraněnou proměnnou i se upraví množina přijímacích stavů F, neboť v abecedě $\sum_{i-1}$ se vyskytuje sekvence vedoucích nul dvakrát. Po následném převodu na DKA se to samo neprojeví, je třeba dodělat algoritmus, který to zařídí. Jsou dvě možnosti - první zařídí upravení množiny přijímacích stavů ještě před samotným nedeterministickým hledáním řešení prozkoumáním všech stavů a přechodů do přijímacích stavů skrze nově vytvořený znak posloupnosti samých nul. Druhá možnost je po konečném převodu na DKA prozkoumat stavy, které nejsou v množině přijímacích stavů, zda z nich je dosažitelný přijímací stav libovolně dlouhou sekvencí vedoucích nul. Pokud ano, je tento původně nepřijímací stav do této množiny přidán.

Práce s logickými operátory $\wedge$ a $\vee$ již byla popsána výše (provede se průnik a sjednocení). Úprava implikace a ekvivalence je taktéž snadná: $A \Rightarrow B$ se upraví na: $\neg A \vee B$, ekvivalence je oboustranná implikace: $A \Leftrightarrow B$: $\neg A \vee B \wedge \neg B \vee A$.

Pokud je výskyt proměnné ve formuli vázán existenčním kvantifikátorem (existuje x takové, že…), můžeme se tohoto kvantifikátoru zbavit nedeterministickým hledáním řešení pro tuto proměnnou. Při výskytu proměnných vázaných na univerzální kvantifikátor (pro všechna x platí, že…) lze převést tuto formuli na ekvivalentní formuli s použitím pouze existenčních kvantifikátorů následujícím způsobem: $\forall x\, P(x) = \neg\big(\exists x\, \neg P(x)\big)$ [2], [6]. Provede se tedy nejprve doplněk, nedeterministicky se vyhledá řešení pro x a na výsledný automat se opět provede doplněk. Pokud se ve formuli vyskytují pouze proměnné, výsledný DKA bude prázdné slovo. Pokud přijme toto prázdné slovo, tak o celé formuli můžeme říct, že je pravdivá. V opačném případě není.

V případě, že se ve formuli vyskytují volné proměnné (nejsou vázány kvantifikátorem), je naším cílem najít takové hodnoty, pro které bude výsledný DKA přijímat prázdné slovo. Budeme tedy za ně dosazovat hodnoty – nedeterministicky hledat řešení, které výsledný DKA přijme. Můžeme tedy opět nedeterministicky hledat řešení pro tyto proměnné jejich postupným odstraňováním, dokud abeceda nebude opět prázdné slovo.

Ovšem ve formuli se můžou vyskytovat i číselné konstanty. Po krocích, které vedou k odstranění vázaných i volných proměnných v abecedě stále zbývá binární reprezentace těchto konstant. Zde stačí pouhým převedením těchto konstant na jejich binární reprezentaci vytvořit vstupní slovo (postup popsán již na začátku této kapitoly) a rozhodnout o pravdivosti.

Pomocí výše uvedených postupů lze vyhodnotit jak formule pouze s proměnnými, čísly z oboru $\mathbb{N}_0 = \{0,1,2,3, \dots\}$, tak i jejich libovolné kombinace.

3. Implementace programu

Po úvodním nahlédnutí do základů teorie automatů, zkonstruování základních automatů a popisu operací nad těmito automaty nyní popíšu další z bodů zadání bakalářské práce, kterým je implementace vlastního programu. V tomto programu jsou použity výše zmíněné postupy a několik dalších algoritmů pro práci s DKA, jejichž odůvodnění pro nasazení vyplynulo při implementaci tohoto programu.

3.1 Výběr jazyka, rozdělení implementace na jednotlivé části

Pro implementaci jsem se rozhodl využít jazyk Java z důvodu nejvíce osobních zkušeností s tímto jazykem oproti ostatním jazykům, tudíž pro mě i celkově snadnější implementaci.

Celý program se dělí na 3 základní části - načtení vstupní formule, operace spojené s převedením této formule na DKA a vyhodnocení vstupní formule, zda patří do jazyka rozpoznávaného tímto automatem. Na vstupu je aritmetická formule, požadovaný výstup je hodnota z dvouprvkové množiny hodnot {TRUE, FALSE} říkající, zda je tato formule pravdivá či nikoliv.

3.2 Načítání formule

Vstupní formuli je třeba nejprve načíst. Formule se načítají ze vstupního souboru, který je zadán jako parametr při spuštění programu. Jsou zapsány v textovém souboru, každá na novém řádku. Soubor je tedy načítán po řádcích a dále zpracováván. Načtení se skládá ze 2 základních částí – lexikální a syntaktické analýzy.

3.2.1 Lexikální analyzátor

Úlohou lexikálního analyzátoru [7] neboli scanneru, je rozdělit vstupní formuli na posloupnost jednotlivých lexikálních jednotek (čísla, proměnné, kvantifikátory, závorky, operátory). Tyto lexémy jsou reprezentovány ve formě tzv. tokenů, které jsou poté předány syntaktickému analyzátoru k dalšímu zpracování. Rozhodl jsem se pro následující reprezentaci lexémů:

∀	∃	proměnné	konstanty	<	≤	>	≥
V	E	a-z	0,1,2,…	<	<=	>	>=

+	=	¬	∧	∨	⇒	⇔	(,)
+	=	!	&	|	=>	<=>	(,)

Obr. 15: Tabulka použitých lexémů.

Dále jsou použity hranaté závorky, které uvozují logickou formuli.

Práce scanneru vypadá tak, že se stále opakuje cyklus, dokud není rozpoznán některý z definovaných tokenů, nebo nedojde na konec formule (ten je tvořen koncem řádku). Běh tohoto cyklu vypadá následovně:

1. Nejdříve jsou přeskočeny tzv. bílé znaky (mezery a tabulátory), poté se načte další znak.

2. Pokud je načteno písmeno, kontroluje se, jestli odpovídá jednomu z kvantifikátorů (vrací se token PRO_VSECHNY, nebo EXISTUJE odpovídající konkrétnímu kvantifikátoru). Jinak jde o proměnnou. Pokud scanner načte proměnnou, uloží si její název, který poté může syntaktickému analyzátoru na vyžádání vrátit a vrátí se token PROMENNA.

3. V případě že se načte číslo, jsou načítány další znaky tak dlouho, dokud není načten jiný znak než číslo. Opět se uloží hodnota načteného čísla a vrátí se token CISLO.

4. Pokud stále nebyl rozpoznán token, porovná se načtený znak se známými jednoznakovými tokeny (logické operátory, operátor sčítání, závorky). Jestli odpovídá některému z těchto definovaných, je vrácen odpovídající token. Speciálním případem je znak '='. V případě načtení tohoto znaku jsou načítány další znaky kvůli ověření, zda nejde o token implikace. Dalším případem je načtení relačních znaků '<' a '>'. V tomto případě jsou opět kontrolovány další znaky, zda se nejedná o token relací větší nebo rovno, či menší nebo rovno, případně ekvivalence. Opět je vracen odpovídající token, pokud je nalezen.

5. V případě, že nebyl rozpoznán žádný z definovaných tokenů (byl načten neznámý znak), je tento znak přeskočen (ignorován) a načte se další znak.

Scanner se zabývá pouze načítáním vstupní formule. Již nekontroluje, jestli je zapsána správně. To je úlohou syntaktického analyzátoru, popsaného v následující kapitole.

3.2.2 Syntaktický analyzátor

Úlohou syntaktického analyzátoru [7] je analyzovat posloupnost načtených tokenů a určit jejich vzájemnou gramatickou strukturu (zařídit, aby se všechny operace vykonávaly v požadovaném pořadí). Další úlohou je určit, jestli je formule správně sestavena. Je totiž zapotřebí zkontrolovat, jestli každá relační operace má levou i pravou stranu (term), jestli logické operátory konjunkce a disjunkce mají také levou i pravou stranu, jestli jsou formule správně uzavřené v závorkách apod.

Z hlediska sestavení formule je potřeba zkontrolovat její základní strukturu:

- Úplným základem je číslo nebo proměnná.
- TERM je tvořen číslem, proměnnou, nebo libovolně dlouhou řadou jejich součtů.
- Na vyšší úrovni je atomická formule ATOM se strukturou (TERM relační operace TERM)
- Logická formule LOG_FORM má strukturu (ATOM logický operátor ATOM)

Lepší a přesnější vyjádření všech pravidel lze vidět na následující gramatice:

<FORMULE> → ‘V‘ <KVANT_FORM>
| ‘E‘ <KVANT_FORM>
| ‘(‘ <ATOM> ‘)‘
| ‘[‘ <LOG_FORM> ‘]‘
| ‘!‘ <FORMULE>

<KVANT_FORM> → ‘proměnná‘ <FORMULE>

<LOG_FORM> → ‘(‘ <ATOM> ‘)‘ <LOG_FORM1>
| ‘[‘ <LOG_FORM> ‘]‘
| ‘!‘ <LOG_FORM>

<LOG_FORM1> → ‘&‘ ‘(‘ <ATOM>) <LOG_FORM1>
| ‘&‘ ‘(‘ <ATOM>) ‘V‘ | ‘&‘ ‘(‘ <ATOM>) ‘E‘
| ‘ |‘ ‘(‘ <ATOM>) <LOG_FORM1>
| ‘ |‘ ‘(‘ <ATOM>) ‘V‘ | ‘ |‘ ‘(‘ <ATOM>) ‘E‘
| ‘=>‘ ‘(‘<ATOM>) <LOG_FORM1>
| ‘=>‘ ‘(‘<ATOM>) ‘V‘ | ‘=>‘ ‘(‘<ATOM>) ‘E‘
| ‘<=>‘ ‘(‘ <ATOM> ‘)‘ <LOG_FORM1>
| ‘<=>‘ ‘(‘ <ATOM> ‘)‘ ‘V‘ | ‘<=>‘ ‘(‘ <ATOM> ‘)‘ ‘E‘
| ‘!‘ <LOG_FORM1> | ‘!‘ ‘V‘ <KVANT_FORM>
| ‘!‘ ‘E‘ <KVANT_FORM>

<ATOM> → <TERM> <REL_SYMB> <TERM>

<TERM> → ‘proměnná‘

| ‘proměnná‘ ‘+‘ <TERM1>

| ‘číslo‘

| ‘číslo‘ ‘+‘ <TERM1>

<TERM1> → ‘proměnná‘

| ‘proměnná‘ ‘+‘ <TERM2>

| ‘číslo‘

| ‘číslo‘ ‘+‘ <TERM2>

<TERM2> → ‘proměnná‘

| ‘proměnná‘ ‘+‘ <TERM2>

| ‘číslo‘

| ‘číslo‘ ‘+‘ <TERM2>

<REL_SYMB> → ‘ =‘ | ‘<‘ | ‘<=‘ | ‘>‘ | ‘ >=‘

Logická formule je uzavřená v hranatých závorkách kvůli jednoznačnosti při načítání vstupu.

3.3 Reprezentace konečných automatů v programu

Všechny základní DKA jsou již vytvořené (byly ukázány v kapitole 2.3). V programu jsem se je rozhodl reprezentovat následující trojicí:

- množina stavů Q, přechodové funkce $\delta : Q \times \sum \rightarrow Q$ a abeceda $\sum$ jsou tvořeny jedinou tabulkou s x řádky a y sloupci. Pojmenuji ji přechodovou tabulkou $P_{x,y}$. Stavy jsou reprezentovány řádky (indexy těchto řádků), abeceda sloupci (indexy těchto sloupců) a přechodové funkce jsou vlastní tabulka, kdy přechod $q_x \times \sum_y \rightarrow$ x odpovídá právě hodnotě v tabulce na souřadnicích [x, y].
- množina přijímacích stavů $F \subseteq Q$ je reprezentována polem booleovských hodnot, kdy hodnota True odpovídá přijímacímu stavu, hodnota False znamená, že stav přijímací není. Toto pole obsahuje právě tolik prvků, kolik je stavů (řádků přechodové tabulky).
- množinou proměnných $B \subseteq \sum_i$. Tato množina slouží k uchovávání proměnných, se kterými právě automat pracuje. Je totiž zbytečné již od počátku tvořit DKA pro celou abecedu $\sum_i$, když je v případě potřeby můžeme později snadno doplnit. Díky tomuto postupu můžeme zpočátku pracovat s podstatně menšími automaty. Tato množina je uchována jako pole hodnot typu Integer,

kdy položky tohoto pole odkazují na indexy proměnných v tabulce všech proměnných, vyskytujících se ve zpracovávané formuli.

Například DKA pro výraz $(x + y = z)$, zmíněný v kapitole 2.3 je v programu reprezentován následovně:

$P_{x,y} =$

0	1	1	0	1	0	2	1
1	1	1	1	1	1	1	1
1	0	2	1	2	1	1	2

$F =$ [True, False, False]
$B =$ [0, 1, 2] za předpokladu, že tabulka všech proměnných obsahuje záznamy: [‘x’,’y’,’z’].

3.4 Operace nad automaty v programu

V programu jsou použity všechny operace nad konečnými automaty, zmiňované v kapitole 2.4. Navíc v průběhu implementace a testování jsem došel ke zjištění, že by bylo vhodné po každém nedeterministickém hledání řešení zmenšit počet stavů (obzvláště v případech, kdy bylo vytvořeno více nových proměnných, které byly následně odstraněny). Po této operaci se totiž často vyskytují v přechodové tabulce ekvivalentní stavy. To jsou takové, kdy pro 2 až n stavů platí, že $q_{x1}, q_{x2}, \dots q_{xn}$ mají všechny přechodové funkce shodné s přechodovými funkcemi pro $q_{x1} \times \sum \rightarrow q_m$. Tyto stavy jsem sjednotil (odstranil stavy $q_{x2}, \dots q_{xn}$) a přepsal jsem v tabulce stavy $q_{x2}, \dots q_{xn}$ na stav q_{x1}. Tímto krokem sice nedošlo k moc velkému snížení počtu stavů (u velkých tabulek se mnohdy nejednalo ani o dvacetinu stavů). Ale v konečném důsledku, kdy je tento DKA se sjednocenými stavy dále upravován, obzvláště když jsou dále odstraňovány proměnné, se již tato úspora stavů násobí. Při testování, kdy jsem nechal zpracovat vstupní soubor s 20 vstupy, došlo ke snížení času potřebného pro zpracování celého vstupu z původních 64 sekund na 35.

3.4.1 Doplněk

Provedení doplňku v programu nebylo těžké. Jedná se o jeden cyklus, ve kterém se projde tabulka, ve které jsou uloženy booleovské hodnoty pro přijímací stavy a v ní se tyto hodnoty znegují.

3.4.2 Průnik a sjednocení

Operace průniku a sjednocení jsou implementovány v jednom algoritmu. Jsou totiž totožné, pouze na konci je rozdíl v označení přijímacích stavů. Na vstupu jsou 2 DKA, pro které se bude provádět požadovaná operace, která je určena druhým parametrem. Algoritmus pracuje takto:

1. Nejprve jsou vzájemně doplněny chybějící proměnné v obou DKA pomocí algoritmu popsaného v následující kapitole.

2. Je založena mezitabulka pro mapování nových stavů s prvním záznamem 0,0. Dále je založena nová přechodová tabulka, na souřadnice [0,0] je zapsána hodnota 0.

3. Je zpracován první řádek průniku obou DKA, vždy když je vytvořen nový stav, který ještě není v mezitabulce, je do ní přidán. Do přechodové tabulky je zapsána pozice tohoto stavu v mezitabulce.

4. Je zpracován zbytek stavů z mezitabulky cyklem pro všechny záznamy. Pro každý stav se v cyklu projdou všechny sloupce přechodové tabulky. Cyklus má tyto kroky:

    - Jsou vyhledány hodnoty v přechodových tabulkách obou DKA s řádky odpovídající dvojici z mezitabulky a sloupci odpovídajícími aktuálním krokům.

    - Pokud se nalezená dvojice přechodů v mezitabulce ještě nevyskytuje, je do ní zapsána.

    - Do přechodové tabulky je uložena pozice vytvořeného stavu v mezitabulce.

5. Je vytvořeno pole přijímacích stavů podle parametru zadaného na vstupu:

    - Při průniku musí být oba stavy z dvojice přijímací

    - Při sjednocení stačí, aby alespoň jeden stav z dvojice byl přijímací.

Je vrácen nový DKA s novým seznamem přijímacích stavů, novou přechodovou tabulkou a seznamem proměnných, který byl vytvořen vzájemným doplněním z obou automatů.

3.4.3 Rozšíření abecedy DKA

Algoritmus pro rozšíření abecedy (doplnění proměnných) má 2 vstupní parametry – DKA, který bude upravován (DKA_1) a DKA, ze kterého se budou doplňovat proměnné (DKA_2). Postup je následující:

1. Je vytvořen vektor hodnot typu Integer pro uložení nových proměnných. V cyklu se projde seznam všech proměnných DKA_2 a proměnné nevyskytující mezi proměnnýma DKA_1 jsou přidány do tohoto vektoru.

2. Je vytvořeno pole pro všechny proměnné, nakopírují se do něj proměnné z DKA_1, poté jsou přidány proměnné z vektoru a toto pole je setřízeno

vzestupně. Uloží se indexy proměnných z DKA_1 (je třeba znát pozice původních proměnných kvůli následnému správnému sestavení znaku – čísla).

3. V cyklu pro všechny znaky abecedy (sloupce přechodové tabulky) se provedou tyto kroky:

    - Index aktuálního sloupce je převeden na hodnotu typu String a zepředu je doplněn nulami na délku odpovídající délce znaku abecedy.

    - Podle pořadí v poli indexů je sestaveno číslo z řetězce v předchozím kroku.

    - Do nové přechodové tabulky se nakopíruje do sloupce odpovídajícího aktuálnímu kroku cyklu obsah sloupce původní přechodové tabulky pro číslo přechodu (znak abecedy) sestaveného v předchozím kroku.

4. Je vracen nový DKA s původními přijímacími stavy, novou přechodovou tabulkou a proměnnýma uloženýma v poli nových proměnných

3.4.4 Nedeterministické hledání řešení

Parametry jsou DKA a proměnná, pro kterou se bude nedeterministicky hledat řešení v tomto DKA. Celý algoritmus se skládá ze 3 základních kroků – odstranění proměnné z abecedy a následném nalezení shodných sloupců v přechodové tabulce, převodu vzniklého NKA na DKA a konečném upravení seznamu přijímacích stavů. Poté je ještě algoritmem pro odstranění ekvivalentních stavů zmenšena přechodová tabulka a je vrácen nový DKA.

Odstranění proměnné se provede tím způsobem, že se nalezne a odstraní řádek, který odpovídá odstraňované proměnné. To se projeví na abecedě tím způsobem, že se rozdělí na určitý počet skupin, který je roven *n*-té mocnině čísla 2, kde *n* je číslo odstraněného řádku. Tyto skupiny jsou totožné, obsahují stejné části abecedy, se kterou automat pracuje. Tímto lze snadno zjistit, které dvojice sloupců přechodové tabulky si budou shodné. Vytvoří se tabulka pro shodné dvojice o dvou řádcích a počtu sloupců odpovídajícím polovině sloupců původní přechodové tabulky. Poté se vytvoří dvojice vždy ze dvou vzájemně sousedících skupin tím způsobem, že prvky těchto dvojic jsou vždy čísla na stejných indexech v těchto skupinách.

S takto vytvořeným seznamem dvojic se následně vytvoří přechodová tabulka NKA, ve které každá dvojice stavů odpovídá dvojici sloupců nalezené v předchozím kroku. Tento NKA se algoritmem pro převod NKA na DKA (popsán v následující kapitole) převede zpět na DKA. Nakonec se provede odstranění ekvivalentních stavů (popsáno v kapitole 3.4.6). Algoritmus končí a takto upravený DKA je vrácen.

3.4.5 Převod NKA na DKA

Na vstupu je NKA. Není to však přímo NKA, protože každý stav přechází pro každý znak právě do 2 stavů, přičemž tyto stavy můžou být shodné. Je to dáno vytvořením přechodové tabulky, které je popsáno v předchozí kapitole. Již se totiž neřeší, jestli jsou prvky každé dvojice shodné, nebo ne. Ale říkejme tomuto NKA s touhle zvláštní tabulkou NKA. Postup převodu na DKA je pak následující:

1. Nejprve je založena hashovací mapa dvojic <Integer, Vector> nazvaná "mezitabulka", kdy klíč bude číslo nového stavu a hodnota vektor se seznamem stavů, do kterých tento stav přechází.

2. Je vytvořen vektor <int []> s názvem "dkaTabulka" pro přechodovou tabulku DKA, který tento algoritmus vytváří a pole int [] "stavDka" pro jednotlivé stavy – řádky této tabulky.

3. Do mezitabulky je vložen nový záznam s klíčem 0 a hodnotou 0 (stav číslo 0, s přechodem do stavu 0).

4. První řádek přechodové tabulky se zpracuje následujícím cyklem:

   - Vytvoří se nový vektor <Integer> s názvem "prechody" pro uložení stavů, do kterých tento stav pro daný znak přechází, vloží se první prvek z dvojice. Zkontroluje se, zda již Vector obsahuje druhý prvek z dvojice. Pokud ne, je do něj vložen.

   - Vektor „prechody" je setřízen vzestupně a provede se kontrola, zda se již vyskytuje v mezitabulce.

     i. Pokud ne, je do mezitabulky vložen nový záznam s klíčem odpovídajícím počtu stavů a hodnotou odpovídající tomuto vektoru. Dále je do pole „stavDka" s indexem odpovídajícím zpracovávanému znaku uložena hodnota odpovídající velikosti mezitabulky (byl do ní vložen nový stav).

     ii. V případě, že se tento stav v mezitabulce již vyskytuje, je nalezena pozice na které se nachází a tato pozice je zapsána do pole „stavDka".

5. Do vektoru „dkaTabulka" je vložen první řádek vytvořený v předchozím kroku.

6. Následující cyklus probíhá, dokud nejsou zpracovány všechny stavy zapsány v mezitabulce:

   - Z mezitabulky je získána hodnota aktuálně zpracovávaného stavu – vektor stavů, do kterých tento stav přechází. Je vytvořeno pole „stavDka" typu Integer pro uložení řádku přechodové tabulky pro

zpracovávaný stav. Pro každý znak (sloupec) přechodové tabulky probíhá následující cyklus:

i. Je vytvořen vektor „prechody“ pro ukládání stavů, do kterých stav přechází přes aktuálně zpracovávaný znak.

ii. Do vektoru „prechody“ jsou z dvojice přechodové tabulky pro NKA odpovídající všem stavům nalezeným v mezitabulce pro zpracovávaný stav uloženy ty stavy, které se v něm ještě nevyskytují.

iii. Vektor „prechody“ je setřízen

iv. Pokud se vektor „prechody“ ještě nenachází v mezitabulce, je do ní zapsán, do pole „stavDka“ na aktuální pozici je zapsána hodnota odpovídající počtu záznamů mezitabulky.

v. V případě že se již v mezitabulce nachází, je vyhledána pozice na které se nachází a tato pozice je zapsána do pole „stavDka“ na aktuální pozici.

- Do vektoru „dkaTabulka“ je zapsáno pole přechodů, vytvořeno v předchozím cyklu.

7. Jsou zpracovány přijímací stavy tím způsobem, že pokud se některý ze stavů, do kterých přechází zpracovávaný stav nachází v původní množině přijímacích stavů, je tento stav označen jako přijímací.

8. V posledním kroku je vektor pro přechodovou tabulku převeden na pole a je vracen nový DKA s nově zpracovanými přijímacími stavy, novou tabulkou a původním seznamem proměnných z předaného NKA.

3.4.6 Odstranění ekvivalentních stavů

Cyklus pro odstranění (sjednocení) ekvivalentních stavů probíhá, dokud se v DKA vyskytují ekvivalentní stavy. Běh tohoto cyklu vypadá následovně:

1. Je vytvořena tabulka „noveStavy“ pro nové pojmenování stavů (číslo odstraněného stavu bude přepsáno na číslo, se kterým je tento stav ekvivalentní)

2. Je vytvořena tabulka booleovských hodnot se dvěma záznamy pro každý stav. První znamená, jestli už byl navštíven, druhá zda zůstane zachován (nebyl k němu nalezen žádný ekvivalentní stav).

3. Všechny stavy jsou zpracovány v cyklu následujícím způsobem:

- Projdou se všechny zbylé stavy a pokud mají společně s kontrolovaným stavem shodnou hodnotu v tabulce přijímacích stavů (oba jsou buď přijímací, nebo nejsou přijímací) a nebyly ještě navštíveny, zkontrolují se všechny jejich přechody. Kontrolují se pouze ty stavy, u kterých je hodnota pro zachování nastavena na True.

- Pokud jsou kontrolované stavy shodné, zapíše se u nalezeného shodného stavu že byl navštíven a hodnota pro zachování je nastavena na False. Dále se do tabulky „noveStavy“ zapíše nové pojmenování stavu, který přijde odstranit.

4. Spočítá se počet stavů, u kterých je hodnota pro zachování rovna True.

5. Čísla stavů jsou sníženy o hodnotu rovnající se počtu stavů, které byly před tímto stavem odstraněny.

6. Je vytvořena nová přechodová tabulka pro stavy, které zůstanou zachovány a nová tabulka přijímacích stavů.

7. Stavy, které zůstávají zachovány jsou zkopírovány do nové přechodové tabulky. Je vyhledána hodnota v původní přechodové tabulce a jí je přiřazena hodnota z tabulky „noveStavy“. Zároveň je v tabulce přijímacích stavů přiřazena odpovídající hodnota z původní tabulky přijímacích stavů.

8. Je vrácen DKA s novou tabulkou přijímacích stavů, novou přechodovou tabulkou a původním seznamem proměnných.

3.5 Uchovávání seznamu proměnných a konstant ve formuli

Pro uchování informací o proměnných a konstantách vyskytujících se ve formuli jsem se rozhodl vytvořit pole hodnot typu String, ve kterém jsou uloženy. Pokaždé když je při zpracovávání načtena proměnná nebo konstanta, kontroluje se, zda se již nachází v tomto seznamu. Pokud ne, je přidána. Proměnné jsou uloženy společně s konstantami, ty lze později v případě potřeby snadno převést zpět na hodnotu typu Integer. Souhrnně je tento seznam označován jako seznam proměnných, když obsahuje i konstanty. Každý automat v programu má pak jako jeden z parametrů pole hodnot typu Integer, odkazující na indexy z tohoto seznamu.

3.6 Rozhodnutí o pravdivosti formule

Vstupem tohoto algoritmu je výsledný DKA se seznamem volných proměnných a konstant (pokud se ve formuli vykytují, jinak je tento seznam prázdný) pro celou formuli, výstupem je booleovská hodnota říkající, zda tento automat přijímá formuli, pro kterou byl sestaven. Algoritmus je složen z těchto kroků:

1. Pokud je abeceda DKA prázdná množina, je přečten stav, do kterého přechází počáteční stav. Je-li tento stav přijímací, je pravdivá i formule, pro kterou byl tento DKA sestaven. V opačném případě je vracena hodnota False. Po tomto kroku algoritmus končí.

2. Pokud se ve formuli vyskytují volné proměnné, je pro ně nedeterministicky hledáno řešení – postupně jsou všechny odstraněny. Po jejich odstranění se zkontroluje, zda je abeceda prázdná množina. Pokud ano, opakuje se krok č. 1.

3. V opačném případě se ve formuli vyskytují ještě konstanty. Postup řešení je pak následující:

    - Je nalezeno největší číslo mezi konstantami a je určena délka jeho binární reprezentace kvůli určení délky tabulky pro převod konstant na abecedu přijímanou automatem – binární reprezentaci těchto čísel. Tato délka je zvětšena o jedničku – pro vložení vedoucí nuly.

    - Provede se naplnění tabulky bitovou reprezentací těchto konstant. Ty jsou zarovnány vpravo a zleva doplněny nulami, aby všechny měly stejnou délku. Tím vznikne vstupní slovo pro automat.

    - Výsledný DKA projde tuto tabulku zprava doleva a po přečtení posledního znaku se vrací hodnota ze seznamu přijímacích stavů pro aktuální stav, čili stav ve kterém skončil DKA po přečtení vstupu.

Tímto algoritmem končí zpracovávání vstupní formule.

3.7 Ukázka řešení konkrétního příkladu

V této kapitole ukážu postup, jak je zpracována vstupní formule $\forall x(x+1>x)$. Nejprve se provede úprava na tvar: $\neg\exists x\neg\exists x_0\neg[(x_0 > x) \wedge (x+1=x_0)]$. Poté se sestrojí 2 základní DKA, první pro relaci $(x_0 > x)$ a druhý pro relaci $(x+1=x_0)$:

x_0	**0**	**0**	**1**	**1**
x	**0**	**1**	**0**	**1**
→0	0	1	2	0
1	1	1	2	1
←2	2	1	2	2

x	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
1	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
x_0	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
↔0	0	1	1	0	1	0	2	1
1	1	1	1	1	1	1	1	1
2	1	0	2	1	2	1	1	2

Obr. 16: Řešený příklad_1

Abecedu DKA pro výraz $(x_0 > x)$ je třeba doplnit o chybějící řádek pro konstantu 1. Oba DKA již pracují se stejnou abecedou, lze tedy provést jejich průnik:

x	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
1	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
x_0	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
0	0	2	0	2	1	0	1	0
1	1	2	1	2	1	1	1	1
←2	2	2	2	2	1	2	1	2

x	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
1	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
~~x_0~~	**~~0~~**	**~~1~~**	**~~0~~**	**~~1~~**	**~~0~~**	**~~1~~**	**~~0~~**	**~~1~~**
→0	0	1	2	3	4	0	5	2
1	1	1	1	1	4	1	4	1
2	2	1	2	1	4	2	4	2
←3	3	1	1	3	4	3	5	1
4	4	1	4	1	4	4	4	4
5	4	3	5	1	5	4	4	5

Obr. 17: Řešený příklad_2

Vlevo je první DKA s doplněnou abecedou, vpravo výsledek průniku. U automatu je naznačen další krok – odstranění vytvořené proměnné x_0, kterou byl nahrazen levý term v původní formuli. Jde vidět, že po tomto odstranění lze ve stavu č. 5 přejít v druhém sloupci do stavu 3, který je přijímací. Po odstranění x_0 se tak rozšiřuje množina přijímacích stavů o stav č. 5. Prozatím se tato změna ještě neprojeví, tato úprava je prováděna až po následném převodu na DKA. Po tomto odstranění vzniká NKA:

x	**0**	**0**	**1**	**1**
1	**0**	**1**	**0**	**1**
→0	0, 1	2, 3	0, 4	2, 5
0,1	0, 1	1 ,2 ,3	0, 1, 4	1, 2, 4, 5
←2,3	1,2,3	1, 2 ,3	2, 3, 4	1, 2, 4, 5
0,4	0, 1, 4	1, 2, 3, 4	0, 4	2, 4, 5
2,5	1, 2, 3, 4	1, 2, 5	2, 4, 5	2, 4, 5
←1,2,3	1, 2, 3	1, 2, 3	1, 2, 3, 4	1, 2, 4, 5
0,1,4	0, 1, 4	1, 2, 3, 4	0, 1, 4	1, 2, 4, 5
1,2,4,5	1, 2, 3, 4	1, 2, 4, 5	1, 2, 4, 5	1, 2, 4, 5
←2,3,4	1, 2, 3, 4	1, 2, 3, 4	2, 3, 4	1, 2, 4, 5
←1,2,3,4	1, 2, 3, 4	1, 2, 3, 4	1, 2, 3, 4	1, 2, 4, 5
2,4,5	1, 2, 3, 4	1, 2, 4, 5	2 ,4, 5	2, 4, 5
1,2,5	1, 2, 3, 4	1, 2, 5	1, 2, 4, 5	1, 2, 4, 5

Obr. 18: Řešený příklad_3

V následujícím kroku je tento NKA převeden na DKA, je rozšířena množina přijímacích stavů, proveden doplněk a bude odstraněna proměnná x:

x	**0**	**0**	**1**	**1**
1	**0**	**1**	**0**	**1**
→0	1	2	3	4
1	1	5	6	7
←2	5	5	8	7
3	6	9	3	10
←4	9	11	10	10
←5	5	5	9	7
6	6	9	6	7
←7	9	7	7	7
←8	9	9	8	7
←9	9	9	9	7
←10	9	7	10	10
←11	9	11	7	7

~~**x**~~	~~**0**~~	~~**0**~~	~~**1**~~	~~**1**~~
1	**0**	**1**	**0**	**1**
↔0	1	2	3	4
←1	1	5	6	7
2	5	5	8	7
←3	6	9	3	10
4	9	11	10	10
5	5	5	9	7
←6	6	9	6	7
7	9	7	7	7
8	9	9	8	7
9	9	9	9	7
10	9	7	10	10
11	9	11	7	7

Obr. 19: Řešený příklad_4

Po odstranění proměnné x vzniká NKA, který je převeden na DKA a opět je proveden doplněk:

1	**0**	**1**
→0	1, 3	2, 4
←1, 3	1, 3 ,6	5, 7, 9, 10
2, 4	5, 8, 9, 10	5, 7, 10, 11
←1, 3, 6	1, 3, 6	5, 7, 9, 10
5, 7, 9, 10	5, 7, 9, 10	5, 7, 9, 10
5, 8, 9, 10	5, 8, 9, 10	5, 7, 9, 10
5, 7, 10, 11	5, 7, 9, 10	5, 7, 10, 11

1	**0**	**1**
→0	1	2
1	3	4
←2	5	6
3	3	4
←4	4	4
←5	5	4
←6	4	6

Obr. 20: Řešený příklad_5

V abecedě je již jen konstanta, dosadí se tedy za konstantu její binární reprezentace a necháme DKA toto slovo přečíst. Jde vidět, že po dosazení 1 a přečtení tohoto vstupu skončí DKA v přijímacím stavu. Což se shoduje se skutečností, že $\forall x(x + 1 > x)$. Pokud by byla místo 1 hodnota 0, již by tento vstup přijat nebyl. Což je správně, neboť neplatí tento vztah pro všechna x v oboru $\mathbb{N}_0 = \{0,1,2,3, \dots\}$.

3.8 Testování programu

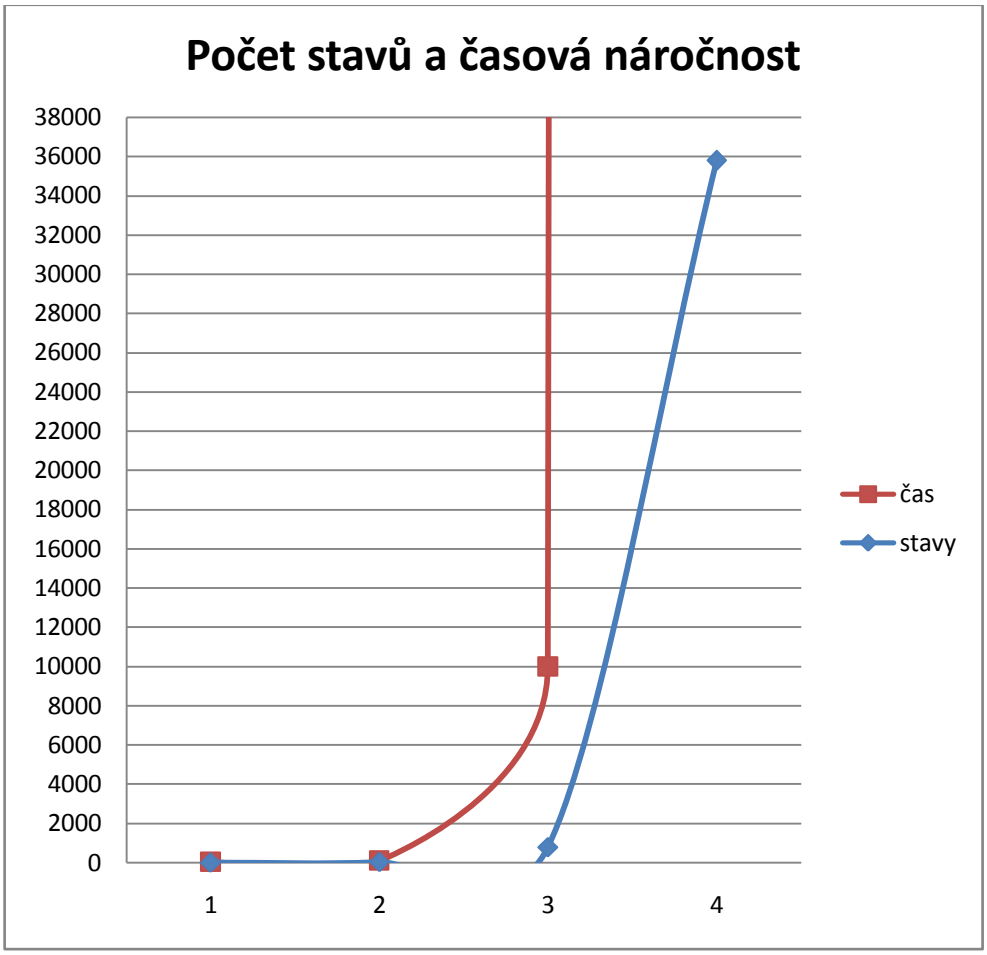
Pro ukázku uvedu výsledek testování 11 formulí. Vstup byl načítán standardním způsobem ze souboru, max. počet stavů uvádí maximální počet stavů, kterého bylo dosaženo v průběhu zpracovávání formule. Sloupec převody NKA → DKA uvádí, kolik bylo v průběhu zpracovávání použito operací převodu NKA na DKA. Čas je v milisekundách, není-li uvedeno jinak. Testování probíhalo na mém notebooku s CPU Intel® Celeron® M 420, 1.6 GHz.

formule	max. počet stavů	převody NKA → DKA	čas
V x (x = x)	1	0	31
V x (x + 1 > x)	12	2	47
V x E y [(x = y+y) | (x = y + y + z)]	147	6	219
V x [(x + 0 = x) & (10 + 10 = 20)]	9	3	47
V x E y E z [(x = y + z) | (x = y + y) | (x = z + z)]	24	6	94
V a V d E c E f [(a + 3 = c) <=> (d + 2 = f)]	16	6	109
V x E y (x + 1 + z < y)	115	5	375
(1 + 2 = 3)	7	1	31
(1 + 2 + 3 = 6)	36	2	109
(1 + 2 + 3 + 4 = 10)	801	3	10 sekund
(1 + 2 + 3 + 4 + 5 = 15)	35 796	4	4 hodiny

Obr. 21: Testování programu

Jak lze vyčíst z tabulky, nezáleží pouze na délce vyhodnocované formule. Záleží hlavně na počtu operací převodu z NKA na DKA a především na velikosti přechodové tabulky převáděného NKA. Doba zpracování „běžné“ formule je řádově ve stovkách milisekund, ale nevinně vypadající (1 + 2 + 3 + 4 = 10) se zpracovává už 10 sekund. Pokud je ve formuli o jeden součet více, jak tomu je v posledním příkladu, doba potřebná ke zpracování se už protáhne na 4 hodiny! Testovat vstup s 5 součty v jednom termu na obyčejném počítači je již nemyslitelné.

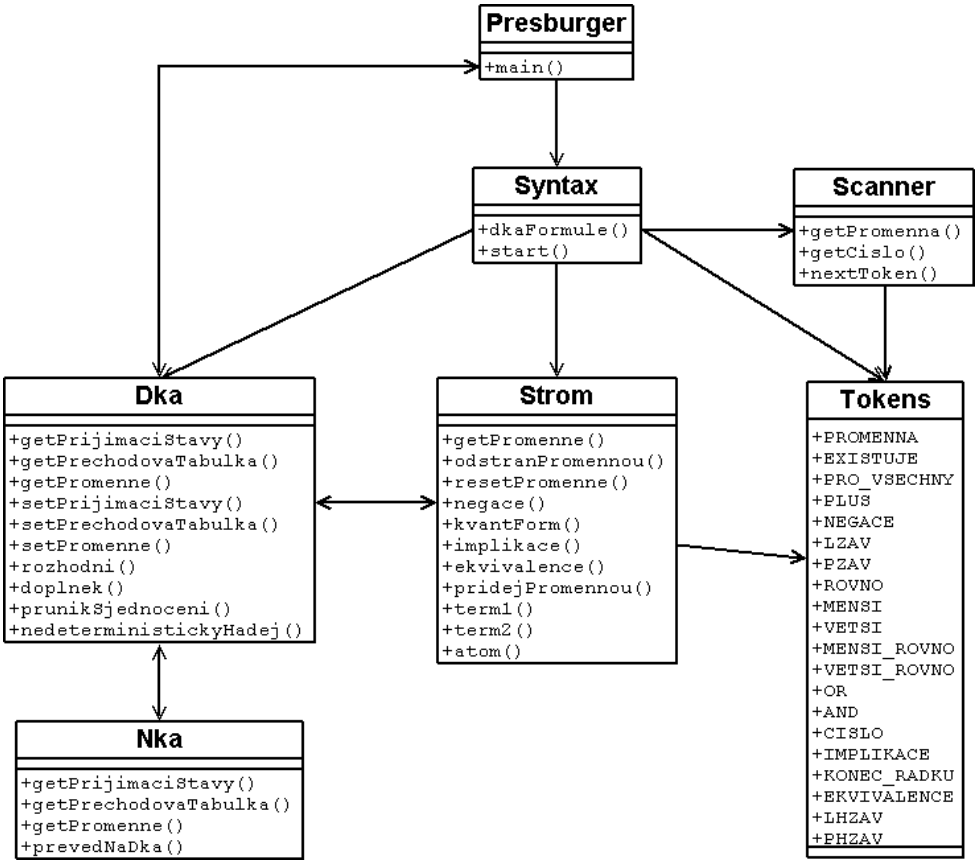
Dalším měřením jsem zjistil následující: u formule s *n* proměnnými přechází průměrně každý stav NKA převáděného na DKA pro každý znak přibližně do *n* stavů. U poslední formule, kde má přechodová tabulka 35 796 stavů má tedy přechodová tabulka celkem: $35\,796 \times 64 \times 6$ záznamů, což je rovno 13 745 664 přechodů celkem.



Obr. 22: Počet stavů a časová náročnost

Graf názorně ilustruje exponenciální nárůst počtu stavů a doby potřebné ke zpracování přechodové tabulky, zejména od termu se 3 součty výše. Osa x značí počet operací součtu v termu, osa y slouží zároveň pro zobrazení počtu stavů a doby v milisekundách.

3.9 Třídní diagram



Obr. 23: Třídní diagram

3.10 Pokyny k programu

Program je spouštěn se dvěma parametry – vstupním a výstupním souborem. Příklad spuštění přes příkazovou řádku: `java Presburger Vstup.txt Vystup.txt.` Ve vstupním souboru jsou zapsány formule ke zpracování, každá na jednom řádku. Do výstupního souboru jsou poté tyto formule zapisovány a k nim je připsána hodnota určující jejich pravdivost (True nebo False). Atomické formule musí být vždy uzavřeny v kulatých závorkách, logická formule je uzavřena v hranatých závorkách. Příklady formulí:

```
(x + y = z)

[(3 + 2 = 5) & (1 + 10 = 11) ]

Vx Ey[(x > y) | (x < y)]

Vx (x + 1 > x)

Vx Ey Ez (x + z + 1 = y)
```

4. Závěr

Byl vytvořen program, který převede vstupní formuli na DKA a určí, zda je tato formule pravdivá v oboru $\mathbb{N}_0 = \{0,1,2,3, \dots\}$. Výsledkem je vždy DKA, díky kterému lze určit pravdivost formule, pro kterou byl zkonstruován. U formule pouze s vázanými či volnými proměnnými stačí nechat tento DKA přečíst prázdné slovo a v jediném kroku rozhodnout o pravdivosti. Pokud se ve formuli vyskytují konstanty, jsou převedeny na jejich binární reprezentaci a přečteny tímto automatem. Pokud je toto slovo přijato, je formule pravdivá, v opačném případě ne.

Doba potřebná k převodu vstupní formule na výsledný DKA je závislá na počtů operací nad automaty prováděnými v průběhu převodu. Kritické jsou zejména termy, ve kterých se vyskytují 3 a více součtů a operace, kdy se nedeterministicky hledá řešení pro určitou proměnnou. Každý průnik automatů může vést až ke zdvojnásobení počtů stavů, nedeterministické hledání řešení a následný převod zpět na DKA opět znásobuje počet stavů. Je tedy možno využít konečných automatů k rozhodování pravdivosti formulí Presburgerovy aritmetiky, je ale třeba zohlednit dobu, která bude potřebná k dosažení výsledku. Pro „jednoduché“ formule typu $\forall x(x + 1 > x)$ můžeme ještě mluvit o časech řádově ve stovkách milisekund, avšak zpracování nevinně vyhlížející formule $(u + v + w + x + y = z)$ si už na běžném stolním počítači vyžádá čas okolo 4 hodin.

Bylo by možno upravit základní DKA, na kterých je tato práce postavena tím způsobem, že by program byl schopný pracovat v oboru celých čísel $\mathbb{Z}$. Dále by bylo možné využít DKA pro vyhledání řešení zadané formule pomocí vyhledání nejkratší cesty z počátečního stavu do nejbližšího přijímacího stavu. Samozřejmě, před krokem vedoucím k odstranění proměnných z abecedy tohoto DKA. Ze slova vytvořeného nalezenou cestou by pak zpětným převodem na konstanty bylo možno zjistit číselné konstanty, jejichž dosazením by byla splněna pravdivost vstupní formule.

Literatura

[1] Petr Jančar, *Studijní opora k předmětům teoretické informatiky, 2004*

[2] *Michael Sipser, Introduction to the Theory of Computation*, PWS Pub. Co., ISBN: 053494728X, 1996

[3] *Petr Jančar, Martin Kot, Zdeněk Sawa, Teoretická informatika* - učební text, Ostrava, 2007

[4] Pierre Wolper, *Bernard Boigelot, An Automata-Theoretic Approach To Presburger Arithmetic Constraints (Extended Abstract), Springer-Verlag, 1995*

[5] *Marek Patrice, Přednášky Západočeské univerzity v Plzni, Přednáška č. 2: Operace nad automaty, nedeterministické automaty, 2009*
URL: `http://home.zcu.cz/~patrke/WWW-KMA/ZTI/ZTI_02_nedeterministicke_konecne_automaty.pdf`

[6] *Ganesh Gopalakrishnan, Computation Engineering : Applied Automata Theory and Logic*, Springer Science +Business Media LLC, ISBN 0-387-24418-2, 2006

[7] *Miroslav Beneš, Překladače – Skripta k předmětu PJP,*
URL:`http://www.cs.vsb.cz/behalek/vyuka/pjp/skripta/skr-mb.pdf`

Přílohy

A. CD se zdrojovými soubory a textem práce